

XQuery

Jef Wijsen

February 26, 2026

```
<?xml version="1.0"?>
<!--The name of this file is catalog.xml-->
<catalog>
    <car>
        <model>Renault CLI0</model>
        <color>blue</color>
        <price unit="BEF">115000</price></car>
    <bike>
        <height>56</height>
        <price unit="EUR">500</price></bike>
    <go-kart>
        <price unit="BEF">3000</price></go-kart>
    <car>
        <model>Peugeot Partner</model>
        <color>red</color>
        <price unit="EUR">12000</price></car>
</catalog>
```

```
doc("catalog.xml")//car[color="blue"]/price
```

≈→

```
<price unit="BEF">115000</price>
```

XQuery query

<seq-expr>

The expression (3,4,5,6,7)
results in {(3,4,5,6,7)}

The range expression 3 to 7
also generates {3 to 7}

There is no notion of nested sequences: (3 to 4,((5,6),7))
also generates {(3 to 4,((5,6),7))}

Sequences are not limited to atomic values:

(<i>{{(1,<two/>,3)}}</i>,4 to 7)
generates {(<i> {(1,<two/>,3)} </i>,4 to 7)}

</seq-expr>

~> the answer

<seq-expr>

The expression (3,4,5,6,7)
results in 3 4 5 6 7

The range expression 3 to 7
also generates 3 4 5 6 7

There is no notion of nested sequences: (3 to 4,((5,6),7))
also generates 3 4 5 6 7

Sequences are not limited to atomic values:

(<i>{(1,<two/>,3)}</i>,4 to 7)
generates <i>1<two/>3</i>4 5 6 7

</seq-expr>

3.9.3.1 Computed Element Constructors

[157] CompElemConstructor ::= "element" (EQName | ("{" Expr "") EnclosedContentExpr

[218] EQName ::= QName | URIQualifiedName

[158] EnclosedContentExpr ::= EnclosedExpr

[36] EnclosedExpr ::= "{" Expr? ""

```
let $bef:=1000000
return
(
<price unit="BEF"> { $bef } </price>,
<price unit="CHF"> { round($bef div 26.0019) } </price>,
element price { attribute unit {"EUR"}, round($bef div 40.3399) }
)
```

~>

```
<price unit="BEF">1000000</price>
<price unit="CHF">38459</price>
<price unit="EUR">24789</price>
```

```
let $bef:=1000000, $p:="prix"
```

```
return
```

```
<convertisseur>
```

```
{ element {$p} { attribute unit {"BEF"}, $bef } }
```

```
{ element {$p} { attribute unit {"CHF"}, round($bef div 26.0019) } }
```

```
{ element {$p} { attribute unit {"EUR"}, round($bef div 40.3399) } }
```

```
</convertisseur>
```

~→

```
<convertisseur>
```

```
  <prix unit="BEF">1000000</prix>
```

```
  <prix unit="CHF">38459</prix>
```

```
  <prix unit="EUR">24789</prix>
```

```
</convertisseur>
```

<comparison>

$(3)=3$ is $\{(3)=3\}$.

$(3,4,5)=(5,6,7)$ is $\{(3,4,5)=(5,6,7)\}$.

$()=()$ is $\{()=()\}$!

$(\langle i \rangle 3 \langle /i \rangle)=(3)$ is $\{(\langle i \rangle 3 \langle /i \rangle)=(3)\}$ due to atomization.

</comparison>

~>

<comparison>

$(3)=3$ is true.

$(3,4,5)=(5,6,7)$ is true.

$()=()$ is false!

$(\langle i \rangle 3 \langle /i \rangle)=(3)$ is true due to atomization.

</comparison>

```
<quantified>
{every $x in 1 to 10 satisfies
  (some $y in 0 to 5, $z in 1 to 5 satisfies $x = $y + $z)}
</quantified>
```

~>

```
<quantified>
  true
</quantified>
```

```
<cond>{  
    if    ( empty(doc("catalog.xml"))//price[@unit="BEF"]) )  
    then "all prices converted"  
    else "still some old prices"  
}</cond>
```

XQuery FLWOR Expressions

For selects a sequence of nodes

Let binds a sequence to a variable

Where filters the nodes

Order by sorts the nodes

Return what to return (gets evaluated once for every node)

```
for    $i in 1 to 3
for    $j in 1 to $i
return <row>{$i}:{$j}</row>
```

~>

```
<row>1:1</row>
<row>2:1</row>
<row>2:2</row>
<row>3:1</row>
<row>3:2</row>
<row>3:3</row>
```

```
for    $i in 1 to 9
let    $j := 1 to $i
return <row>{$i}:{$j}</row>
```

~>

```
<row>1:1</row>
<row>2:1 2</row>
<row>3:1 2 3</row>
<row>4:1 2 3 4</row>
<row>5:1 2 3 4 5</row>
<row>6:1 2 3 4 5 6</row>
<row>7:1 2 3 4 5 6 7</row>
<row>8:1 2 3 4 5 6 7 8</row>
<row>9:1 2 3 4 5 6 7 8 9</row>
```

```

<goldbach>{
let $up := 10
let $primes :=
  (for $i in 2 to 2*$up
    where not(some $j in 2 to $i, $k in 2 to $i satisfies $i=$j*$k)
    return $i)
for $i in 1 to $up
for $j in $primes, $k in $primes
where 2*$i = $j + $k and $j <= $k
return <i>{2*$i} = {$j} + {$k}</i>
}</goldbach>

```

~>

```

<goldbach>
  <i>4 = 2 + 2</i>    <i>6 = 3 + 3</i>    <i>8 = 3 + 5</i>    <i>10 = 3 + 7</i>
  <i>10 = 5 + 5</i>   <i>12 = 5 + 7</i>   <i>14 = 3 + 11</i> <i>14 = 7 + 7</i>
  <i>16 = 3 + 13</i> <i>16 = 5 + 11</i> <i>18 = 5 + 13</i> <i>18 = 7 + 11</i>
  <i>20 = 3 + 17</i> <i>20 = 7 + 13</i>
</goldbach>

```

```
<?xml version="1.0" standalone="no" ?>
<!--The name of this file is emp.xml-->
<employees>
  <emp>
    <name>Ed</name>
    <sal>100</sal>
    <company>UMONS</company>
  </emp>
  <emp>
    <name>Tim</name>
    <sal>50</sal>
    <company>UCL</company>
  </emp>
  <emp>
    <name>An</name>
    <sal>75</sal>
    <company>ULB</company>
  </emp>
</employees>
```

```
for $a in doc("emp.xml")/*emp
where number($a/sal) gt 60
order by number($a/sal)
return ($a/name, $a/company)
```



```
<name>An</name>
<company>ULB</company>
<name>Ed</name>
<company>UMONS</company>
```

```

<answer>{
for $a in doc("emp.xml")/*emp
where number($a/sal) gt 60
order by number($a/sal)
return <emp>{ $a/name,
               $a/company
            }</emp>
}</answer>

```

~>

```

<answer>
  <emp>
    <name>An</name>
    <company>ULB</company>
  </emp>
  <emp>
    <name>Ed</name>
    <company>UMONS</company>
  </emp>
</answer>

```

For all products that are red or blue, get the Euro price and the color.

```
<blue-red-products>{  
for    $product in doc("catalog.xml")/catalog/*  
where  $product/color=("blue","red")  
return element { name($product) }  
        {    <price unit="EUR">{  
                if ( $product/price/@unit="EUR" )  
                then $product/price/text()  
                else round($product/price div 40.3399)  
        }</price>,  
        $product/color  
    }  
}</blue-red-products>
```

~>

```
<blue-red-products>
  <car>    <price unit="EUR">2851</price>
            <color>blue</color></car>
  <car>    <price unit="EUR">12000</price>
            <color>red</color></car>
</blue-red-products>
```

```
<?xml version="1.0" standalone="no"?>
<!--The name of this file is cy.xml-->
<companies>
  <cy>    <cname>UMONS</cname>
          <loc>Mons</loc>
          <loc>Charleroi</loc></cy>
  <cy>    <cname>UCL</cname>
          <loc>Louvain</loc></cy>
  <cy>    <cname>ULB</cname>
          <loc>Bruxelles</loc></cy>
</companies>
```

Get names of employees who work for a company located in Mons or Brussels; order employees by increasing salary.

```
for $emp in doc("emp.xml")//emp
for $cy in doc("cy.xml")//cy
where $emp/company = $cy/cname
and $cy/loc = ("Mons","Bruxelles")
order by number($emp/sal)
return $emp/name
```

~> `<name>An</name>`
`<name>Ed</name>`

```
for $emp in doc("emp.xml")//emp
for $cy in doc("cy.xml")//cy[cname=$emp/company]
where $cy/loc="Mons" or $cy/loc="Bruxelles"
order by number($emp/sal)
return $emp/name
```

Who earns the highest salary?

```
let $emp := doc("emp.xml")//emp
for $a in $emp
where $a/sal = max($emp/sal)
return $a/name
```

~> <name>Ed</name>

What is the most expensive product?

```
declare namespace my="my.uri";
declare function my:toBEF ($p as element(price)) as element(price)
{
  if ($p/@unit="BEF") then $p
  else if ($p/@unit="EUR") then <price unit="BEF">{40.3399*$p}</price>
  else error("Unknown Unit")
};

<most-expensive-product>{
  let $max := max(doc("catalog.xml")/catalog/*/my:toBEF(price))
  return doc("catalog.xml")/catalog/*[my:toBEF(price) = $max]
}</most-expensive-product>
```

~→

```
<most-expensive-product>  
  <car>  
    <model>Peugeot Partner</model>  
    <color>red</color>  
    <price unit="EUR">12000</price>  
  </car>  
</most-expensive-product>
```

```
declare namespace my="my.uri";
declare function my:toBEF ($p as element(price)) as element(price)
{
  if ($p/@unit="BEF")
  then $p
  else if ($p/@unit="EUR")
    then <price unit="BEF">{round(40.3399*$p)}</price>
    else <price>?</price>
};
```

```

<HTML><BODY>
<H1>Cars</H1>
<TABLE BORDER="3">
<TR><TH>Car Model</TH><TH>Price</TH></TR>
{ for $car in doc("catalog.xml")//car
  return <TR><TD> { $car/model/text() } </TD>
          <TD> { ($car/my:toBEF(price))/text() } </TD></TR> }
</TABLE>
<H1>Bikes</H1>
<TABLE BORDER="3">
<TR><TH>Frame Height</TH><TH>Price</TH></TR>
{ for $bike in doc("catalog.xml")//bike
  return <TR><TD> { $bike/height/text() } </TD>
          <TD> { ($bike/my:toBEF(price))/text() } </TD></TR> }
</TABLE>
</BODY></HTML>

```

XQuery is a **Functional Programming Language** (and is Turing Complete)

- The value of a variable in a functional program never changes once defined. That is, variables are immutable.

For example, computing the sum $\sum_{i=1}^n i$. The following pseudocode is **not** functional (and **not** XQuery):

```
n := 100
sum := 0
for i = 1 to n
    sum := sum + i
return sum
```

In XQuery:

```
let $n:=100
return sum(1 to $n)
```

Plus d'exemples

Un fleuriste livre des bouquets à la maison.

Chaque espèce de fleur (tulipe, rose. . .) a un prix exprimé en centimes d'euro. Par exemple, le prix d'une rose est de 150 centimes d'euro, indépendamment de sa couleur; voir la ligne

```
<fleur fnom="rose" prix="150"/>
```

Un bouquet rassemble des fleurs de différentes espèces et couleurs. Par exemple, le bouquet *Exotique* est composé d'un seul tournesol et trois iris bleus. Voir les lignes :

```
<bouquet bnom="Exotique">  
  <fleur fnom="tournesol" nombre="1"/>  
  <fleur fnom="iris" couleur="bleu" nombre="3"/>  
</bouquet>
```

La couleur n'est pas renseignée si la fleur n'existe qu'en une seule couleur (par exemple, tous les **tournesols** sont de même couleur).

```
<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE fleuriste [
  <!ELEMENT fleuriste (fleurs, bouquets)>
  <!ELEMENT fleurs (fleur)*>
  <!ELEMENT bouquets (bouquet)*>
  <!ELEMENT bouquet (fleur)*>
  <!ELEMENT fleur (#PCDATA)>
  <!ATTLIST fleur fnom CDATA #REQUIRED>
  <!ATTLIST fleur prix CDATA #IMPLIED>
  <!ATTLIST fleur couleur CDATA #IMPLIED>
  <!ATTLIST fleur nombre CDATA #IMPLIED>
  <!ATTLIST bouquet bnom CDATA #REQUIRED>
]>
```

```
<fleuriste>
  <fleurs>
    <!-- Les prix sont en centimes d'euro -->
    <fleur fnom="tulipe" prix="100"/>
    <fleur fnom="rose" prix="150"/>
    <fleur fnom="iris" prix="250"/>
    <fleur fnom="tournesol" prix="300"/>
  </fleurs>
  <bouquets>
    <bouquet bnom="Valentin">
      <fleur fnom="rose" couleur="rouge" nombre="10"/>
    </bouquet>
    <bouquet bnom="Belge">
      <fleur fnom="tulipe" couleur="noir" nombre="3"/>
      <fleur fnom="tulipe" couleur="jaune" nombre="4"/>
      <fleur fnom="tulipe" couleur="rouge" nombre="6"/>
    </bouquet>
    <bouquet bnom="Exotique">
      <fleur fnom="tournesol" nombre="1"/>
      <fleur fnom="iris" couleur="bleu" nombre="3"/>
    </bouquet>
  </bouquets>
</fleuriste>
```

(: Taille des bouquets.:)

```
<bouquets>{  
for $b in //bouquets/bouquet  
let $taille := sum(for $f in $b/fleur return $f/@nombre)  
return  
<bouquet nom="{ $b/@bnom}" taille="{ $taille}" fleurs"/>  
}</bouquets>
```

Note : Après Database > Open and Manage... dans BaseX, ne plus utiliser la fonction doc("...").

~>

```
<bouquets>  
  <bouquet nom="Valentin" taille="10 fleurs"/>  
  <bouquet nom="Belge" taille="13 fleurs"/>  
  <bouquet nom="Exotique" taille="4 fleurs"/>  
</bouquets>
```

Elimination de la variable \$taille

(: Taille des bouquets.:)

```
<bouquets>{  
for $b in //bouquets/bouquet  
return  
<bouquet  
  nom="{ $b/@bnom }"  
  taille="{sum(for $f in $b/fleur return $f/@nombre)} fleurs"/>  
}</bouquets>
```

Encore plus court



```
(: Taille des bouquets.:)
```

```
<bouquets>{  
for $b in //bouquets/bouquet  
return  
<bouquet  
  nom="{ $b/@bnom }"  
  taille="{sum($b/fleur/@nombre)} fleurs"/>  
}</bouquets>
```

```
(: Prix des bouquets.:)
```

```
<bouquets>{  
for $b in //bouquets/bouquet  
let $prixBouquet := sum(  
    for $f in $b/fleur  
    let $prixFleur := //fleurs/fleur[@fnom = $f/@fnom]/@prix  
    return $f/@nombre * $prixFleur)  
return  
<bouquet nom="{ $b/@bnom}" prix="{ $prixBouquet}"/>  
}</bouquets>
```

~>

```
<bouquets>  
  <bouquet nom="Valentin" prix="1500"/>  
  <bouquet nom="Belge" prix="1300"/>  
  <bouquet nom="Exotique" prix="1050"/>  
</bouquets>
```

Elimination de \$prixBouquet

(: Prix des bouquets.:)

```
<bouquets>{  
for $b in //bouquets/bouquet  
return  
<bouquet  
  nom="{ $b/@bnom }"  
  prix="{sum(  
    for $f in $b/fleur  
    let $prixFleur := //fleurs/fleur[@fnom = $f/@fnom]/@prix  
    return $f/@nombre * $prixFleur  
  )  
  }"  
</>  
}</bouquets>
```

Elimination de \$prixFleur

(: Prix des bouquets.:)

```
<bouquets>{  
for $b in //bouquets/bouquet  
return  
<bouquet  
  nom="{ $b/@bnom }"  
  prix="{sum(  
    for $f in $b/fleur  
    return $f/@nombre * //fleurs/fleur[@fnom = $f/@fnom]/@prix  
  )  
  }"  
>  
>  
}</bouquets>
```

L'espèce de fleur la plus chère (sans usage de max)

(: Les fleurs les plus chères. :)

`//fleurs/fleur[not(@prix < //fleurs/fleur/@prix)]/@fnom`

Explication : Soit $A = \{a_1, a_2, \dots, a_n\}$, un ensemble fini. Un élément m de A est l'élément maximal de A si

$$\forall x \in A : m \geq x,$$

ce qui est équivalent à :

$$\neg \exists x \in A : m < x. \quad (1)$$

Suivant la “sémantique existentielle” de XPath, $(m) < (a_1, a_2, \dots, a_n)$ est vrai s'il existe $x \in \{a_1, a_2, \dots, a_n\}$ tel que $m < x$. En conséquence, (1) est exprimé en XPath par `not((m) < (a1, a2, ..., an))`.

Les bouquets contenant une même espèce de fleur en plusieurs couleurs
(sans usage de count)

(: Les bouquets contenant plusieurs couleurs d'une même fleur :)

```
//bouquet[fleur[@fnom=following-sibling::fleur/@fnom]]/@bnom
```

Noter: `following::sibling::` est par rapport à `fleur` qui précède le
crochet `[`. On peut aussi écrire:

```
//bouquet[fleur[following-sibling::fleur/@fnom=@fnom]]/@bnom
```

BaseX

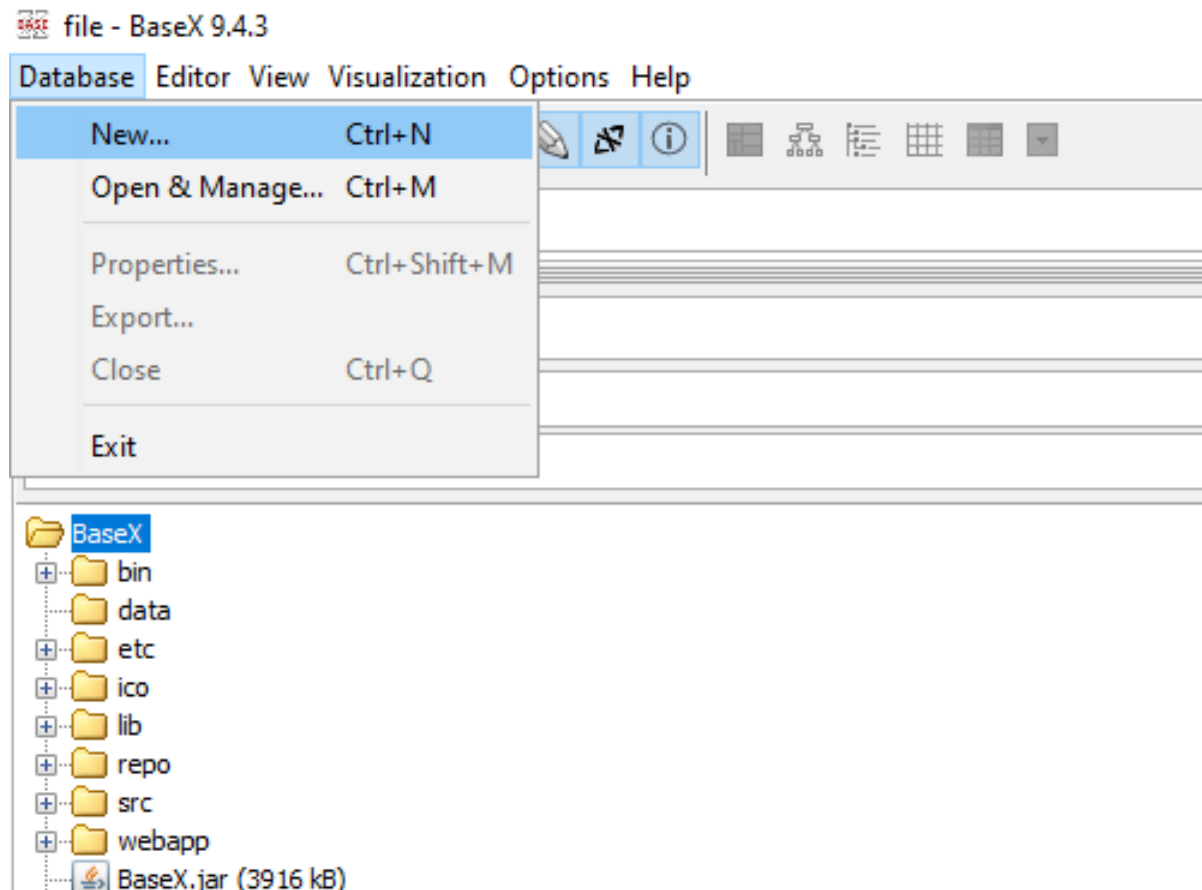
Installing and Running BaseX in Four Steps

Step 1: Download and Install BaseX

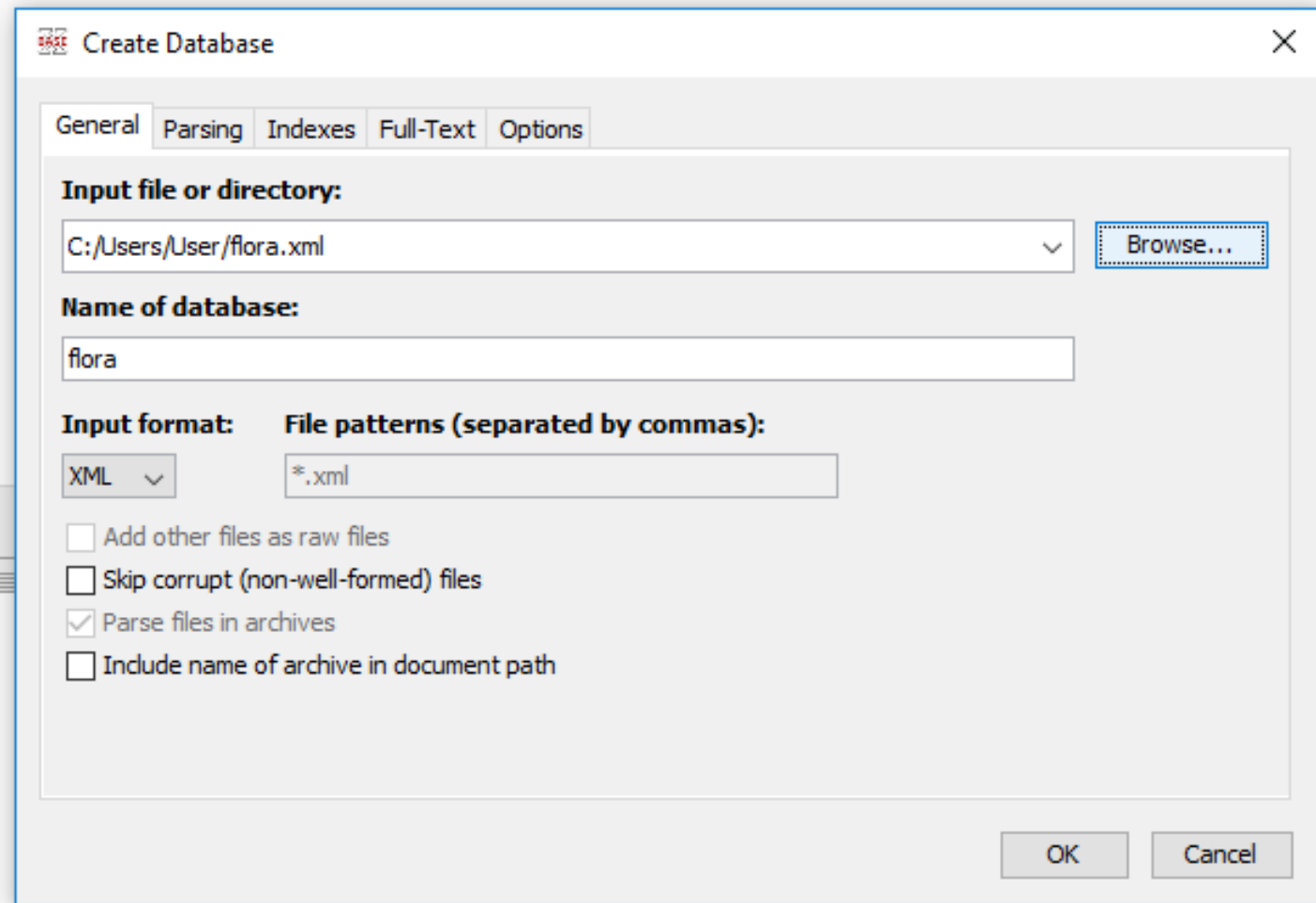
<https://basex.org/>

Step 2: Create a `flora.xml` file.

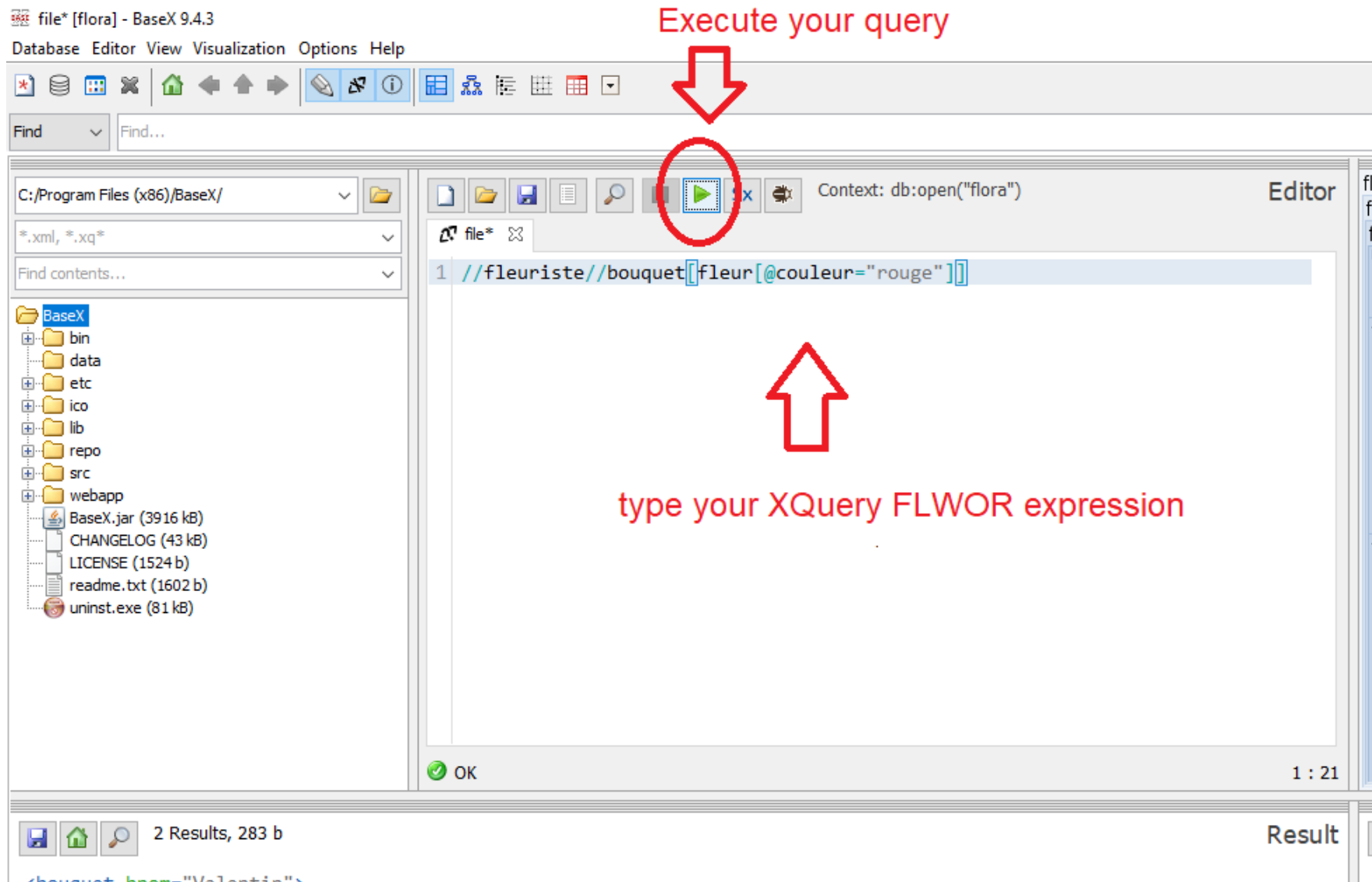
Step 3.1: Create a New Database



Step 3.2: Create a New Database



Step 4: Run an XQuery FLWOR Expression



Questions

1. Quels bouquets n'ont pas de fleurs jaunes?
2. Quels sont les bouquets *monochromes*, i.e., les bouquets où toutes les couleurs renseignées sont identiques? Valentin et Exotique pour notre exemple. Est-ce exprimable sans usage de `count`?
3. Quel est le bouquet de taille maximale? Est-ce exprimable sans usage de `max`?
4. Quel est le prix à payer si l'on achetait tous les bouquets?
5. Quel est le bouquet le plus cher?
6. Renvoyer l'attribut `@bnom` de chaque bouquet qui contient toutes les espèces de fleur. Est-ce exprimable sans usage de `"every" | "some" ... "satisfies"`?
7. Renvoyer l'attribut `@fnom` de chaque fleur qui ne fait partie d'aucun bouquet.
8. Renvoyer l'attribut `@fnom` de chaque fleur qui est présente dans chaque bouquet.
9. Écrire une requête en **XPath** qui renvoie le nom de chaque bouquet qui contient une espèce de fleur qui n'apparaît dans aucun autre bouquet. **Il n'est pas permis d'utiliser des fonctions d'agrégation telles que `count`, `min` et `max`.**

Solutions

1. Quels bouquets n'ont pas de fleurs jaunes?

```
//bouquet[not(fleur/@couleur="jaune")]/@bnom
```

2. Quels sont les bouquets *monochromes*, i.e., les bouquets où toutes les couleurs renseignées sont identiques? Valentin et Exotique pour notre exemple. Est-ce exprimable sans usage de count?

```
//bouquet[count(distinct-values(fleur/@couleur))<=1]/@bnom
```

Noter que **distinct-values** est nécessaire dans le cas suivant:

```
<bouquet bnom="Valentin">  
  <fleur fnom="rose" couleur="rouge" nombre="10"/>  
  <fleur fnom="tulipe" couleur="rouge" nombre="10"/>  
</bouquet>
```

Sans usage de count:

```
//bouquet  
[not(fleur[@couleur!=following-sibling::fleur/@couleur])]/@bnom
```

ou encore: 🤔

```
//bouquet[not(fleur/@couleur!=fleur/@couleur)]/@bnom
```

3. Quel est le bouquet de taille maximale? Est-ce exprimable sans usage de max?

```
let $maxTaille := max(for $b in //bouquet
                      return sum($b/fleur/@nombre))
return //bouquet[sum(fleur/@nombre)=$maxTaille]
```

Elimination de \$maxTaille :

```
//bouquet[sum(fleur/@nombre)=max(for $b in //bouquet
                                return sum($b/fleur/@nombre))]
```

`return` doit être précédé de `for` ou `let`, ce qui implique qu'il n'y a plus de `return` après l'élimination de \$maxTaille.

En XPath pur, avec et sans usage de max : 🙌

```
//bouquet[sum(fleur/@nombre)=max(//bouquet/sum(fleur/@nombre))]
```

```
//bouquet[not(sum(fleur/@nombre)<//bouquet/sum(fleur/@nombre))]
```

4. Quel est le prix à payer si l'on achetait tous les bouquets?

```
let $p:= for $b in //bouquets/bouquet
    return sum(for $f in $b/fleur
        let $prixFleur:=
            //fleurs/fleur[@fnom = $f/@fnom]/@prix
        return $f/@nombre * $prixFleur)
return sum($p)
```

Analyse

```
let $p:= for $b in //bouquets/bouquet
    return sum(for $f in $b/fleur
        let $prixFleur:=
            //fleurs/fleur[@fnom = $f/@fnom]/@prix
        return $f/@nombre * $prixFleur)
return sum($p)
```

La sous-requête bleue renvoie les prix des bouquets:

```
1500
1300
1050
```

Plus compact :

```
sum(for $f in //fleurs/fleur  
return sum(//bouquet/fleur[@fnom=$f/@fnom]/(@nombre*$f/@prix)))
```

Explication: Pour chaque fleur (“for \$f in //fleurs/fleur”), on calcule sa contribution (sum) au prix total, puis on fait la somme (sum) pour toutes les fleurs.

5. Quel est le bouquet le plus cher?

```
let $bp:= for $b in //bouquets/bouquet
    let $prixBouquet :=
        sum(for $f in $b/fleur
            let $prixFleur:=
                //fleurs/fleur[@fnom = $f/@fnom]/@prix
            return $f/@nombre * $prixFleur)
        return <bou nom="{ $b/@bnom}" pri="{ $prixBouquet}"/>
return $bp[@pri=max($bp/@pri)]/@nom
```

Analyse

```
let $bp:= for $b in //bouquets/bouquet
    let $prixBouquet :=
        sum(for $f in $b/fleur
            let $prixFleur:=
                //fleurs/fleur[@fnom = $f/@fnom]/@prix
            return $f/@nombre * $prixFleur)
    return <bou nom="{ $b/@bnom}" pri="{ $prixBouquet}"/>
return $bp[@pri=max($bp/@pri)]/@nom
```

La [sous-requête bleue](#) renvoie tout nœud <bou nom="x" pri="y"> tel que x est le nom d'un bouquet qui coûte y:

```
<bou nom="Valentin" pri="1500"/>
<bou nom="Belge" pri="1300"/>
<bou nom="Exotique" pri="1050"/>
```

6. Renvoyer l'attribut @bnom de chaque bouquet qui contient toutes les espèces de fleur. Est-ce exprimable sans usage de "every" | "some" ... "satisfies"?

```
//bouquets/bouquet[  
    every $f1 in //fleurs/fleur/@fnom satisfies  
        (some $f2 in fleur/@fnom satisfies($f1 = $f2))  
    ]/@bnom
```

Sans usage de some:

```
//bouquets/bouquet[  
    every $f1 in //fleurs/fleur/@fnom satisfies  
        ($f1=fleur/@fnom)  
    ]/@bnom
```

Noter l'usage de la “sémantique existentielle” de =.

7. Renvoyer l'attribut @fnom de chaque fleur qui ne fait partie d'aucun bouquet.

```
//fleurs//@fnom[not(.=//bouquets//@fnom)]
```

8. Renvoyer l'attribut @fnom de chaque fleur qui est présente dans chaque bouquet.

```
//fleurs//@fnom[every $b in //bouquet satisfies(.= $b/fleur/@fnom)]
```

- Est-ce exprimable en XPath?

9. Renvoyer le nom de chaque bouquet qui contient une espèce de fleur qui n'apparaît dans aucun autre bouquet.

```
//bouquets/bouquet[  
    fleur[not(@fnom=parent::*preceding-sibling::*fleur/@fnom)]  
        [not(@fnom=parent::*following-sibling::*fleur/@fnom)]  
    ]/@bnom
```

- “*” peut être remplacé par “bouquet”.
- Est-ce exprimable en XPath sans usage de l'axe parent?